

Improvement of the Cascadic Multigrid Algorithm with a Gauss Seidel Smoother to Efficiently Compute the Fiedler Vector of a Graph Laplacian

Shivam Gandhi - Tufts University Department of Mathematics *

November 2015

1 Abstract

In this paper, we detail the improvement of the Cascadic Multigrid algorithm with the addition of the Gauss Seidel algorithm in order to compute the Fiedler vector of a graph Laplacian, which is the eigenvector corresponding to the second smallest eigenvalue. This vector has been found to have applications in graph partitioning, particularly in the spectral clustering algorithm. The algorithm is algebraic and employs heavy edge coarsening, which was developed for the first cascadic multigrid algorithm. We present numerical tests that test the algorithm against a variety of matrices of different size and properties. We then test the algorithm on a range of square matrices with uniform properties in order to prove the linear complexity of the algorithm.

2 Introduction

The Fiedler vector has seen numerous applications within computational mathematics, primarily within the fields of graph partitioning and graph drawing [1]. In particular, we require eigenvalues and eigenvectors for a successful run of the spectral clustering algorithm that partitions a network into clusters[2]. Although many languages have built in eigenvalue methods, the spectral clustering method requires a specialized eigenvalue algorithms to account for massive network size. In fact, spectral clustering becomes unfeasible for networks of size over 1000 as computing the eigenvalues through matrix inversion becomes inefficient. Therefore, we require specialized multigrid algorithms to find the eigenvectors and eigenvalues in less than $O(N^3)$ time.

*shivam.jgandhi@gmail.com

The Cascadic Multigrid Algorithm is an effective method for computing the second largest eigenvalue and eigenvector, where the eigenvector is called the Fiedler vector. The main methods for calculating eigenvalues and eigenvectors include the Lanczos method and the power method. However, these methods become unfeasible for large matrices ($|V| > 1000$). Furthermore, many networks can have over 1000 nodes which correlates to a matrix with dimension higher than 1000. For this reason, we require the cascadic multigrid algorithm, as it solves the eigenvalue problem on coarser levels and projects the solution upwards until the solution is projected to the original matrix [3].

When calculating the eigenvalues and eigenvectors of symmetric positive definite matrices more generally, the Jacobi and PCG methods provide good approximations. These can be extended to calculating the Fiedler vector and its eigenvalue [1].

In this paper, we improve upon the previously made Cascadic Multigrid Algorithm by introducing a Gauss Seidel smoother on each level. We employ the previously established heavy edge coarsening that selects the edge with the heaviest weight between two vertices. The refinement procedure continues to use power iteration on a modified matrix. This method does not require the inversion of matrices, unlike Rayleigh-Quotient iteration, thereby making it a much more optimal method. As always, the eigenvector calculated on coarse levels are projected to a finer level with interpolation matrices. The eigenvector that gets calculated and projected up to a higher level serves as the guess for the next Gauss Seidel iteration on the finer level. Finally, on the highest level, we calculate the Rayleigh Quotient to achieve the eigenvalue.

The paper is organized to provide a logical introduction to the algorithm. In section 3, we provide definitions and background knowledge required to understand multigrid methods. We also introduce heavy edge coarsening, power iteration, and the Gauss Seidel method. This culminates in a presentation of the algorithm developed. In section 4, we introduce the numerical tests of the algorithm. We compare the algorithm to the previous cascadic multigrid algorithm and various other multigrid algorithms that are meant to calculate the Fiedler vector. We also compare spectral clustering with the build in eigenvalue calculating function in MATLAB to spectral clustering that employs our algorithm to show efficient. In the final section, we wrap up the paper and discuss future improvements to multigrid algorithms that calculate Fiedler vectors.

3 Modified Cascadic MG Method for Computing the Fiedler Vector

First we formally introduce the concepts of the graph Laplacian and the Fiedler vector. A weighted graph $G = (V, E, w)$ is undirected if the edges are unoriented.

Definition 2.1 $G = (V, E, w)$ is a weighted graph. The Laplacian of G , $L(G) \in \mathbb{R}^{n \times n}$, shortened to L , where $n = |V|$, is denoted as follows

$$L(G)_{(i,j)} = \begin{cases} d_{v_i} & , \text{ if } i = j \\ -w_{(i,j)} & , \text{ if } i \neq j \end{cases}$$

where d_{v_i} is the degree of vertex i and $w_{i,j}$ is the weight of the edge connecting v_i and v_j .

This Laplacian is positive semi-definite and diagonally dominant, and the sum of and row or column of L is zero. This makes the smallest eigenvalue 0 with the corresponding vector $[1, 1, \dots, 1]^T$. We are particularly interested in the second smallest eigenvalue and eigenvector.

Definition 2.2 The second smallest eigenvalue of the Laplacian of a graph G is called the algebraic connectivity. This eigenvalue must be greater than or equal to 0. The corresponding eigenvector ϕ_2 is called the Fiedler vector of G .

The importance of the Fiedler vector is detailed in [4, 5].

It is important to note that the coarsest graph must be very small in size at around $|V| < 25$. A direct power iteration is used at this coarsest level to obtain an eigenvector. Afterwards, the eigenvector is projected upwards and then smoothed using Gauss-Seidel.

We now introduce heavy edge coarsening for our cascadic algorithm. In our algorithm, $L^i \in \mathbb{R}^{n_i \times n_i}$. Heavy edge coarsening is iterated on the graph Laplacian in order to create multiple levels for solving. This algorithm makes up the setup phase.

Algorithm 1 Heavy Edge Coarsening

```

1: procedure HEC(L)
2:    $c \leftarrow 0$ 
3:    $p \leftarrow \text{randperm}(n_i)$ 
4:    $q \leftarrow \text{zeros}(n_i, 1)$ 
5:   for  $i = 1 \rightarrow n_i$  do
6:     if  $q(p(i)) = 0$  then
7:        $m \leftarrow \text{argmin}(L(:, p(i)))$ 
8:       if  $q(m) = 0$  then
9:          $c \leftarrow c + 1$ 
10:         $q(m) = c$ 
11:         $q(p(i)) = c$ 
12:      else
13:         $q(p(i)) = q(m)$ 
14:      end if
15:    end if
16:  end for
17:   $I_i^{i+1} \leftarrow \text{zeros}(c, n_i)$ 
18:  for  $i = 1 \rightarrow n_i$  do
19:     $I_i^{i+1}(q(i), i) = 1$ 
20:  end for
21: end procedure

```

Heavy edge coarsening is further detailed in [3], and several properties of the algorithm are proved as well.

Next, we formally introduce the Gauss Seidel method. This method takes a guess vector and solves a linear system using that guess. In our algorithm, the we use the vector projected upwards from the coarser level as the guess. This was similar to power iteration, as we used the projected vector as the first guess for power iteration as well. The values A and b are the original values in the linear system $Ax = b$. X0 is our initial guess to the solution of this system. N denotes the number of iterations allowed while tol represents the tolerance of error. The algorithm outputs a solution to $Ax = b$ within our denoted error.

Algorithm 2 Gauss Seidel

```

1: procedure G-S(A, b, X0, tol, N)
2:    $k \leftarrow 1$ 
3:   while  $k \leq N$  do
4:     for  $i = 1 \rightarrow n$  do
5:        $x_i = 1/a_{ii}[-\sum_{j=1}^{i-1} (a_{ij}x_j) - \sum_{j=i+1}^n (a_{ij}X0_j) + b_i]$ 
6:       if  $|x - X0| < tol$  then
7:         output  $[x_1, x_2, \dots, x_n]$ 
8:       end if
9:        $k = k + 1$ 
10:    for  $i = 1 \rightarrow n$  do
11:       $X0_j = x_i$ 
12:    end for
13:  end for
14: end while
15:  Output  $[x_1, x_2, \dots, x_n]$ 
16: end procedure

```

We discuss two theorems that confirm that the Gauss Seidel method will converge to a solution in our multigrid algorithm.

Theorem 2.1: The Gauss Seidel method converges if A is symmetric positive definite or if A is strictly or irreducibly diagonally dominant.

Theorem 2.2: Let A be a symmetric positive definite matrix. Then the Gauss-Seidel method converges for any arbitrary choice of initial approximation x .

A proof of these theorems can be found in [6]. All of our graph Laplacians on all levels are symmetric positive definite and diagonally dominant therefore the Gauss Seidel method will converge on all levels.

With our component algorithms defined and sufficiently detailed, we can now outline the procedure for our algorithm. We begin with a setup phase that has heavy edge coarsening set up the levels on which we do computations. After this, we solve the eigenvalue problem on the coarsest level. We then begin projecting our eigenvector upwards and using Gauss Seidel on finer and finer levels until we

get to the finest level, our original matrix. At this level, we use Gauss Seidel one last time to yield the Fiedler vector and then calculate the Rayleigh quotient for the algebraic connectivity. We input the finest level graph Laplacian and the algorithm outputs the Fiedler vector and corresponding eigenvalue.

Algorithm 3 Gauss Seidel Cascadic Multigrid

```

1: procedure STEP 1: SETUP PHASE( $L$ )
2:    $i = 0$ 
3:   while  $n_i > 25$  do
4:      $I_i^{i+1} \leftarrow HEC(L^i)$ 
5:      $L^{i+1} = I_i^{i+1} L^i (I_i^{i+1})^T$ 
6:      $i = i + 1$ 
7:   end while
8:    $J \leftarrow i$ 
9: end procedure
10: procedure STEP 2: COARSEST LEVEL SOLVING PHASE( $L^J$ )
11:    $y(J) \leftarrow GS(L^J, rand(n_J))$ 
12: end procedure
13: procedure STEP 3: CASCADIC REFINEMENT PHASE( $y^J, L$ )
14:   for  $j = J - 1 \rightarrow 0$  do
15:      $y^j = (I_i^{i+1})^T y^{(j+1)}$ 
16:      $y^j \leftarrow GS(L^j, y^j)$ 
17:   end for
18: end procedure

```

Structurally, this algorithm is similar to other multigrid algorithms in that it begins with a setup phase and solves on the coarsest level upwards. It is nearly identical to the Cascadic Multigrid Algorithm with the sole difference being in the Gauss Seidel replacing power iteration.

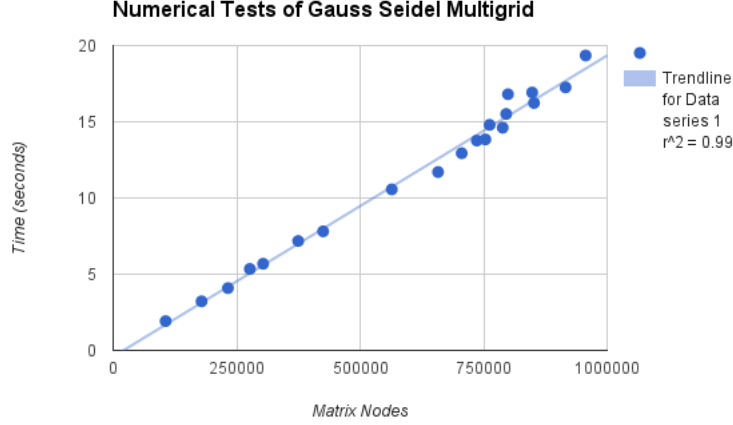
4 Numerical Tests

We perform numerical tests on a variety of graphs listed on Table 4.1. The graphs were taken from the University of Florida Sparse Matrix Collection [7]. The computations were performed on an HP Envy with a 2.40 GHz Intel Core i7 Processor with 8.00 GB of RAM. We consider the performance of the Gauss Seidel Cascadic Multigrid Algorithm to matrices with over 8000 nodes. We use a tolerance $(u^k, u^{k-1}) > 1 - 10^{-6}$.

Matrix Name	Matrix Size	Matrix Edges	CGMG runtime (s)
barth5	15606	45878	0.371467
bcsstk32	44609	985046	1.242307
bcsstk33	8738	291583	0.381135
brack2	62631	366559	1.307903
copter1	17222	96921	0.42307
ct2010	67578	168176	1.265944
halfb	224617	6081602	6.694857
srb1	54924	1453614	1.582835
wing_nodal	10937	75488	0.40845

Next, we show that the algorithm is $O(N)$. We run the algorithm on uniform square arrays of various sizes and show that the runtime increases linearly according to the matrix size. The amount of nodes and edges increases linearly therefore we can expect the runtime of the algorithm to also increase linearly. Because multigrid algorithms run in linear time, it is important that the Gauss Seidel smoother does not change the runtime, otherwise it would be an inferior algorithm to use. The r value is very close to 1, indicating that the algorithm does in fact have $O(N)$ complexity.

Matrix Nodes	Time (seconds)
106276	1.921614
178929	3.220836
232324	4.088426
276676	5.344172
303601	5.684314
374544	7.178143
425104	7.811554
564001	10.565033
657721	11.704087
705600	12.936846
736164	13.768696
753424	13.843865
762129	14.799933
788544	14.613115
795664	15.51262
799236	16.808463
848241	16.922279
851929	16.233831
915849	17.257426
956484	19.349795



5 Conclusion

In this paper, we have presented an improvement on the existing Cascadic Multigrid Algorithm by introducing a Gauss Seidel smoother as opposed to a power iteration smoother on each level. The algorithm is effective in calculating the algebraic connectivity and the Fiedler vector and is able to partition graphs quickly.

Having shown that the Gauss-Seidel Cascadic Multigrid Algorithm runs in linear time, we can now discuss its benefits and pitfalls. If our initial graph Laplacian is not sparse, then Gauss Seidel will fail as a smoother since it is inherently meant to work on sparse matrices. In this case, other multigrid algorithms would be optimal. However, the Gauss Seidel smoother works well for most Laplacians as most Laplacians are sparse. Furthermore, we showed that the algorithm is effective in calculating the Fiedler vector of a variety of different graphs.

We see future works modifying the smoother more. Future improvements could include changing the Gauss Seidel to a Lanczos smoother. Krylov subspace methods are costly for calculating the eigenvalues and eigenvectors of large matrices but produce accurate results. Furthermore, future works could include a convergence analysis of the cascadic multigrid algorithm on a more general level and take into account the Gauss Seidel method in the convergence. Of particular interest is our algorithm's convergence with respect to elliptic eigenvalue problems.

6 Acknowledgement

The research presented here was undertaken by Shivam Gandhi and directed by Dr. Xiaozhe Hu of the Tufts University Mathematics Department.

References

- [1] Golub, Gene H.; Van Loan, Charles F. (1996), *Matrix Computations* (3rd ed.), Baltimore: Johns Hopkins, ISBN 978-0-8018-5414-9.
- [2] U. V. Luxburg. A tutorial on spectral clustering. *Statistics and Computing*, (2007), 395–416 (electronic)
- [3] J.C. Urschel, X. Hu, J. Xu, and L. T. Zikatanov. A Cascadic Multigrid Algorithm for Computing The Fiedler Vector of Graph Laplacians. (2014) 1-15 (electronic).
- [4] M. Fiedler, Algebraic connectivity of graphs, *Czechoslovak Math. J.*, **23(98)** (1973), 298–305.
- [5] M. Fiedler, A property of eigenvectors of nonnegative symmetric matrices and its application to graph theory, *Czechoslovak Math. J.*, **25(100)**:4 (1975), 619–633.
- [6] B. N. Datta. *Numerical Linear Algebra and Applications: Second Edition*. SIAM 2010.
- [7] T.A. Davis and Y. Hu, The University of Florida sparse matrix collection, *ACM Trans. Math. Software*, **38**:1 (2011), Art. 1, 25